The LAURA Experiment:
Adapting Code to Use The SSD on the Cray Y-MP

Teresa M. Griffie

Report RND-90-001, November 1990

Numerical Aerodynamic Simulation Division
NASA Ames Research Center

The Fortran program LAURA was modified to reduce use of the Cray Y-MP's main memory during execution by sending arrays to the machine's solid-state storage device. A measurable decrease in executable size resulted from this modest recoding, suggesting that programs currently too large to fit into existing run queues might be altered slightly to reduce their in-core memory requirements. A method was developed for identifying arrays which would be most appropriate for recoding.

I. Introduction

Some scientists with large codes may be unable to take advantage of the speed of the Cray Y-MP because its run queues are too small to accommodate their programs. In order to address this situation, various techniques for reducing in-core memory use were investigated. In this experiment, we modified Peter Gnoffo's LAURA (Langley Aerothermodynamic Upwind Relaxation Algorithm) program to evaluate the benefits of reducing executable size by utilizing the Cray Y-MP solid-state storage device (SSD). The SSD is the Cray's high-speed DRAM-based auxiliary memory, offering transfer rates around 1 Gbit/sec. It is significantly faster than disk I/O or Cray-2 main memory. Executable size was reduced by storing arrays in the SSD and retrieving or rewriting them in small sections, so that less main memory is used during program execution..

LAURA, which calculates hypersonic and thermal flow around a body, was selected for this experiment because it employs large 3- and 4-dimensional arrays that can be stored in the SSD. Most of these arrays, as well as the DO-loops and conditional statements, rely on three parameters which are set within the program itself: IQ, JQ, and KQ.

- IQ is the number of cells from the axis to the boundary of the body. Its minimum value is 10.
- JQ is the circumference index, and determines the number of degrees in a hemisphere for each cell. For example, 10^{o} for each cell means JQ is 18.
- KQ is the shock index, the number of cells into the free stream. Its minimum value is 16.

These values govern the size of the calculation grid, and hence the executable size of the program. In the program's initial form, IQ was equal to 30, JQ was equal to 12, and KQ was equal to 32.

II. Approach

We selected the method for buffering data to the SSD by examining how the arrays would be accessed during program execution. In particular, we noted the order in which data would be required, and the fact that I/O requests to the SSD must be sent in segments of 1 block (512 words). These two criteria determined the size of the 2-dimensional array in memory as well as how I/O to the SSD would be handled for calculations within loops. Since many of the outer loops use KQ, data files stored in the SSD were composed of KQ different 2-dimensional arrays, arranged as they would have been in regular memory. In addition, the 2-dimensional array size had to be an integral multiple of 512 to accommodate the SSD I/O requirements.

Here is an example of the translation from original code to new code:

Original code:

```
      PARAMETER(IQ=30,JQ=12,KQ=32)
      COMMON/BLKA/A(IQ,JQ,KQ),B(IQ,JQ,KQ)
      DO 10 K=1,KQ
      DO 10 J=1,JQ
          DO 10 I=1,IQ
              A(I,J,K)=AINIT
              B(I,J,K)=VOL*A(I,J,K)+VINIT
10    CONTINUE
```

New code:

```
    PARAMETER(IQ=30,JQ=12,KQ=32)
    DIMENSION ARR(32,16),BRR(32,16)
    DO 20 K=1,KQ
    DO 10 J=1,JQ
        DO 10 I=1,IQ
            ARR(I,J)=AINIT
            BRR(I,J)=VOL*ARR(I,J)+VINIT
10  CONTINUE
    CALL GETARR(1,ARR,32,16,K,0)
    CALL GETARR(2,BRR,32,16,K,0)
20 CONTINUE
    SUBROUTINE GETARR(IUNIT,ARRR,I,J,K,IRW)
    DIMENSION ARRR(I,J)
    INDEX=I*J*(K-1)
    CALL SETPOS(IUNIT,INDEX)
    IF (IRW.EQ.1)BUFFER IN (IUNIT,1)(ARRR(1,1),ARRR(I,J))
    IF (IRW.EQ.0)BUFFER OUT (IUNIT,1)(ARRR(1,1),ARRR(I,J))
    RETURN
    END
```

The key feature of the new code is the subroutine GETARR. This routine positions the data file at the appropriate location using SETPOS on file IUNIT. It then performs a read or a write on the file, depending on flag IRW, using unformatted I/O with BUFFER IN and BUFFER OUT.

III. Results

Initially 40 arrays of the form A(IQ,JQ,KQ) were chosen to be written to the SSD. Over the course of several trials, we observed a marked decrease in the in-core memory use (e.g., 96 MW to 69 MW in one case). However, at the same time, the CPU time used increased significantly due to large amounts of SSD I/O. For example, with the parameters IQ = 24, JQ = 12, and KQ = 32, forty iterations of the modified LAURA code used 150.18 user CPU seconds and 364.23 system CPU seconds. This was in contrast to the 26.68 user CPU seconds and 0.41 system CPU seconds clocked by the original version. The large CPU overhead associated with this technique led us to further refine our methods to achieve a better balance between memory use and CPU use.

First, we examined several other arrays which represented a major portion of the in-core memory usage. These arrays have the form ARR(IQ*JQ,4,2). Unfortunately, they were used in different ways than the A(IQ,JQ,KQ) arrays, and so were not good candidates for the GETARR subroutine. Next, we analyzed in-line functions to see if they would be more appropriate for these arrays, but the additional amount of I/O required would offset any memory reduction. One array, however, was found to fit into both the above group and the original group: It has the form ARR(IQ*JQ,16,16,KQ). We wrote a new subroutine GETRRR, which is similar to GETARR, for this array.

By altering this one array, substantial memory reductions were achieved, even for modest values of IQ, JQ, and KQ. For the same example, (IQ=24, JQ=12, KQ=32), code size went from 5.21 MW with original code to 3.00 MW with new code, yielding a 42% memory reduction. Keeping the same IQ and JQ, memory requirements were reduced by 53% for KQ = 64, and 60% for KQ = 128.

Tables 1, 2, and 3 list the results for different sizes and parameters. Figures 1, 2, and 3 show selected results graphically.

IQ JQ	New size. in MW	Original size in MW	Percentage Decrease in memory	New time, user & system (in seconds)	Original time, user & system (in seconds)
24 12	3.00	5.21	42	28.87 1.26	28.68 0.41
24 18	4.17	7.48	44	40.61 1.64	40.50 0.31
24 24	5.33	9.75	45	51.98 2.62	51.86 0.47
24 36	7.67	14.30	46	76.09 2.80	75.79 0.45
30 12	3.60	6.36	43	34.45 1.57	34.28 0.62
30 18	5.05	9.19	45	49.13 1.97	48.72 0.56
30 24	6.50	12.03	46	63.61 3.09	63.41 0.50
30 36	9.40	17.69	47	92.95 3.46	91.75 0.56
40 12	4.59	8.28	45	43.91 1.94	43.84 0.54
40 18	6.52	12.05	46	63.19 3.60	63.03 0.64
40 24	8.44	15.81	47	81.73 3.39	81.32 0.52
40 36	12.30	23.35	47	120.28 4.65	119.84 1.44
60 12	6.58	12.11	46	62.78 2.43	62.71 0.84
60 18	9.46	17.75	47	90.59 3.25	90.28 1.19
60 24	12.33	23.39	47	119.31 4.01	119.66 1.33
60 36	18.09	34.67	48	175.30 7.79	175.10 1.87

Table 1. Comparison of in-core memory sizes and user and system time for 40 iterations for KQ=32.

IQ JQ	New size. in MW	Original size in MW	Percentage Decrease in memory	New time, user & system (in seconds)	Original time. user & system (in seconds)
24 12	4.08	8.65	53	28.85 1.56	28.88 0.28
24 18	5.77	12.62	54	40.59 2.64	40.46 0.97
24 24	7.46	16.60	55	52.13 2.67	51.90 1.11
24 36	10.84	24.55	56	76.36 3.94	75.79 2.43
30 12	4.96	10.65	54	34.42 1.91	34.32 0.34
30 18	7.04	15.61	55	48.96 2.56	48.78 0.84
30 24	9.15	20.57	56	63.69 3.05	63.44 1.61
30 36	13.36	30.49	56	92.13 4.14	92.40 1.95
40 12	6.38	13.99	54	43.93 2.81	43.77 1.64
40 18	9.17	20.60	55	63.22 3.41	63.00 0.79
40 24	11.97	27.20	56	81.80 4.34	81.37 1.75
40 36	17.55	40.40	57	120.51 5.31	119.74 2.68
60 12	9.25	20.68	55	62.90 3.09	62.55 1.40
60 18	13.42	30.57	56	90.85 4.10	90.13 2.33
60 24	17.60	40.45	56	119.36 5.80	118.13 2.76
60 36	25.94	60.22	57	176.37 8.40	173.45 2.05

Table 2. Comparison of in-core memory sizes and user and system time for 40 iterations for KQ = 64

IQ JQ	New size. in MW	Original size in MW	Percentage Decrease in memory	New time, user & system (in seconds)	Original time. user & system (in seconds)
24 12	6.24	15.52	60	28.84 1.64	28.61 0.83
24 18	8.97	22.91	61	40.62 2.23	40.44 1.21
24 24	11.71	30.29	61	52.16 2.58	52.07 1.21
24 36	17.18	45.05	62	76.42 4.42	75.71 1.87
30 12	7.63	19.24	60	34.47 2.35	34.34 1.04
30 18	11.04	28.45	61	49.02 2.48	48.79 0.90
30 24	14.45	37.67	62	63.74 3.04	63.53 1.49
30 36	21.26	56.09	62	92.63 6.33	92.10 2.48
40 12	9.95	25.43	61	43.93 2.15	43.66 0.87
40 18	14.48	37.70	62	63.33 2.98	62.99 1.35
40 24	19.01	50.00	62	82.14 3.16	81.55 1.26
40 36	28.06	74.51	62	121.09 6.82	*
60 12	14.59	37.81	61	62.95 3.12	62.54 1.21
60 18	21.36	56.19	62	90.93 4.82	90.29 1.72
60 24	28.13	74.57	62	119.74 6.02	*
60 36	41.66	111.33	63	176.76 12.73	*

* Executable too large to run on Reynolds

Table 3. Comparison of in-core memory sizes and user and system time for 40 iterations for KQ = 128

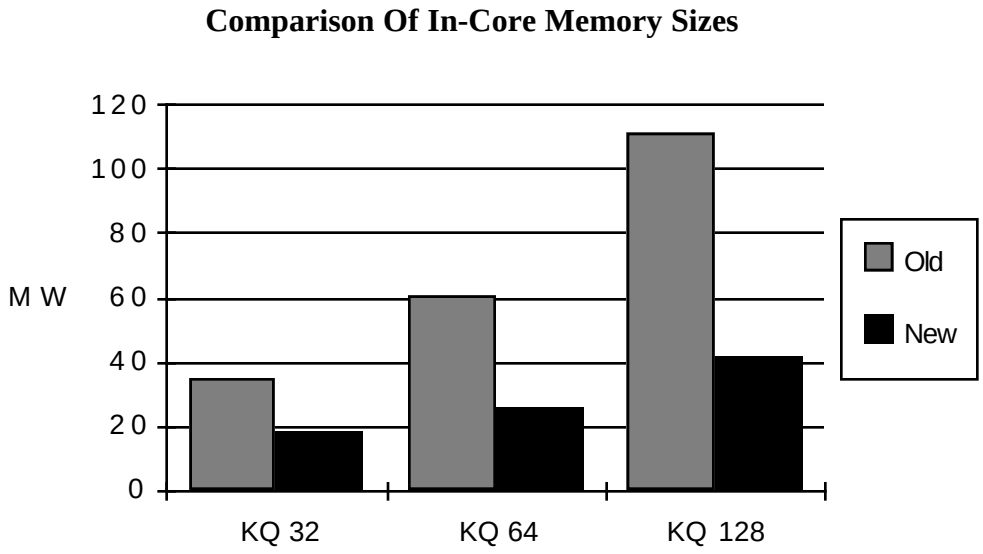

Figure 1. Comparison of In-Core memory size before and after re-writing code to run arrays on the SSD. In this test data set, IQ=60 and JQ=36.

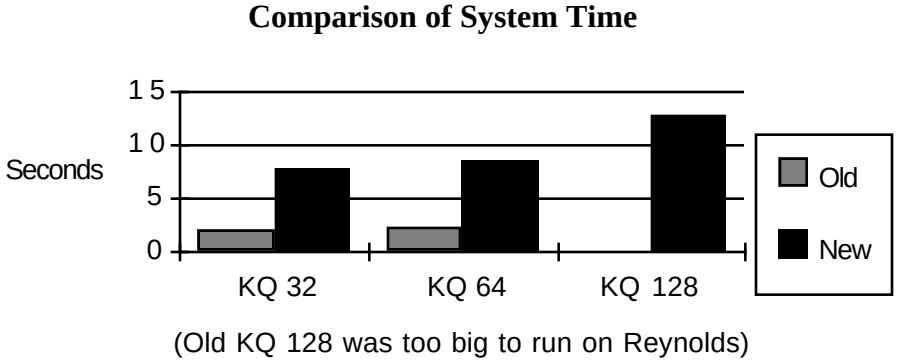

Figure 2. Comparison of system time before and after re-writing code to run arrays on the SSD. In this test data set, IQ=60 and JQ=36.

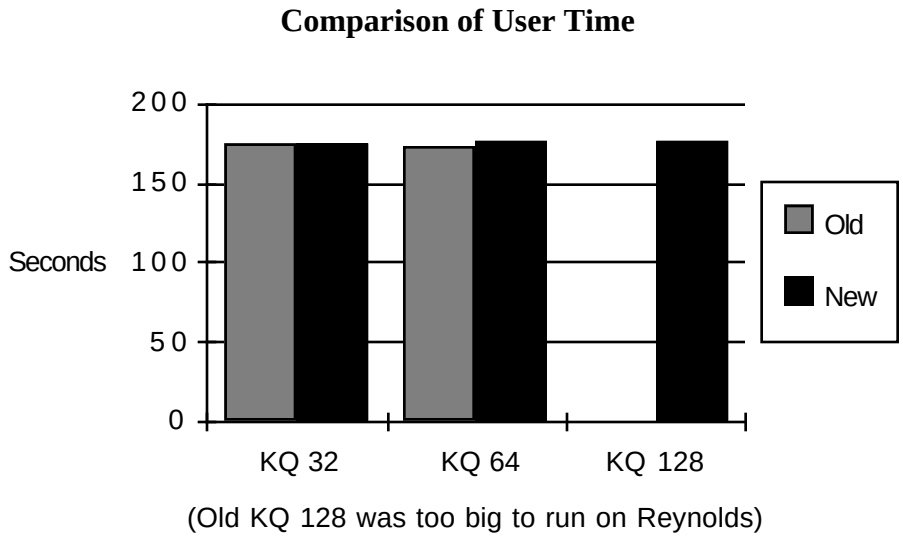

Figure 2. Comparison of user time before and after re-writing code to run arrays on the SSD. In this test data set, IQ=60 and JQ=36.

IV. Size differences

We used the /bin/size utility resident on the Cray Y-MP to determine and compare actual executable sizes because some of the original programs sizes were greater than the largest queue sizes currently available on the machine. The /bin/size utility yields a size which, on the average, is only 1% less than the actual runtime size of the code.

Tables I, II, and III, shown earlier, compare memory size and performance between the original and modified versions of LAURA over various values of the three parameters spanning the domain representing actual physical problems.

The different values of the parameters were:

IQ = 24, 30, 40, and 60
JQ = 12, 18, 24, and 36
KQ = 32, 64, and 128.

It was found that the memory size differences between original and new versions of LAURA for each test case directly related to the size of KQ relative to the other parameters. When KQ is much larger than IQ and JQ, memory reduction is much greater than when KQ is approximately the same as IQ or JQ. This occurs because memory required by the original array is decreased by an amount proportional to KQ when writing KQ new arrays out to the SSD.

V. Runtime differences

The major difference in run time will be in system CPU time, due to SSD I/O. How much CPU time the system actually spends reading and writing to the SSD is directly proportional to KQ, and to the number of reads and writes in the actual code, which is low. In this case, since only one array is being stored in the SSD, the difference in total CPU time is small. The slight difference in user CPU time is due to subroutine calls to GETRRR in the new version. The difference in system CPU time is due to the system setting up data transfer to the SSD; this involves many different functions and is influenced by such factors as the current job load, number of swapped jobs, cache status, etc. Although several tests were run for various amounts of data to establish I/O rates to the SSD, it was not possible to quantitatively determine how dependent the SSD I/O overhead is on the different parameters which affect it. This issue should be the subject of future studies.

VI. Conclusions

The LAURA experiment demonstrates that a measurable decrease in executable size can result from a modest amount of recoding. This suggests that users with codes that are currently too large to fit into existing run queues can still benefit from the Y-MP's speed by moving selected arrays in their code to the SSD.

User CPU time increases associated with this method are due to additional calls to the subroutine which performs the SSD I/O. By ensuring that this routine is small and efficient, or perhaps by using an in-line function instead, any increase in user CPU time should be minimal. Increases in system CPU time will vary somewhat with the amount of data being written to the SSD and with the system utilities that handle the actual transfers. Therefore, the key to achieving a marked decrease in the in-core memory size without significantly increasing overall CPU time is to select arrays that:

- Are large relative to in-core memory size;
- Utilize a structure in the code which allows the SSD I/O routine to fit in without major restructuring;
- Are accessed a minimum number of times, to keep the system CPU time low.